Activity: Use regular expressions to find patterns

Introduction

Security analysts often analyze log files, including those that contain information about login attempts. For example as an analyst, you might flag IP addresses that relate to unusual attempts to log in to the system.

Another area of focus in cybersecurity is detecting devices that require updates. Software updates help prevent security issues due to vulnerabilities.

Using regular expressions in Python can help automate the processes involved in both of these areas of cybersecurity. Regular expression patterns and functions can be used to efficiently extract important information from strings and files.

In this lab, you'll write regular expressions to extract information such as device IDs or IP addresses.

Tips for completing this lab

Scenario

In this lab, you're working as a security analyst and your main tasks are as follows:

- extracting device IDs containing certain characters from a log; these characters correspond with a certain operating system that requires an update.
- extracting all IP addresses from a log and then comparing them to those that are flagged in a list.

Task 1

In order to work with regular expressions in Python, start by importing the `re` module. This module contains many functions that will help you work with regular expressions. By running the following code cell, the module will be available through the rest of the notebook.

In [2]:
```
# Import the `re` module in Python

import re
```

Task 2

Currently, you are looking for device IDs that begin with `"r15"` . These characters indicate that the device is running an operating system that must be updated.

You're given a log of device IDs, stored in a variable named `devices` . Your eventual goal is to extract the device IDs that start with the characters `"r15"` . For now, display the contents of the whole string to examine what it contains. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [3]:
```
# Assign `devices` to a string containing device IDs, each device ID represent
ed by alphanumeric characters
devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253
be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac i4l56nq g07h55q 0
81qc9t r159r1u"

# Display the contents of `devices`
print(devices)
```

```
r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253be78 ac742
a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac i4l56nq g07h55q 081qc9t r1
59r1u
```

Hint 1

Task 3

In this task, you'll write a pattern to find devices that start with the character combination of `"r15"` .

Use the regular expression symbols `\w` and `+` to create the pattern, and store it as a string in a variable named `target_pattern` .

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell. Note that the code cell will contain only variable assignments, so running it will not produce an output.

In [5]:
```
# Assign `devices` to a string containing device IDs, each device ID represent
ed by alphanumeric characters

devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253
be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac i4l56nq g07h55q 0
81qc9t r159r1u"

# Assign `target_pattern` to a regular expression pattern for finding device I
Ds that start with "r15"
target_pattern = r"r15\w+"
```

Hint 1

Hint 2

Hint 3

Question 1

What regular expression pattern did you use? For each component of the pattern, what would happen if it were missing?

I used the following regular expression pattern:

CODE I USED FOR REGULAR EXPRESSION: r15\w+

Explanation of each component:

r15: This part matches the literal characters "r15" in the string.

\w+: This part matches one or more word characters (letters, digits, or underscores) that come after the "r15" prefix.

If the "r15" part is missing, the pattern won't match any device IDs. If the \w+ part is missing, the pattern will only match the exact string "r15" followed by nothing, which is likely not the intended behavior for finding device IDs. The combination of both ensures that the pattern matches device IDs starting with "r15" followed by any alphanumeric characters numerical value.

Task 4

Use the `findall()` function from the `re` module to find the device IDs that the `target_pattern` matches with. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

Note: In order to use `re.findall()` in Tasks 4, 7, 8, 9 and 11, you must have previously run the code `import re` in Task 1.

In [6]:
```
# Import the re module
import re

# Define the devices string
devices = "r262c36 67bv8fy 41j1u2e r151dm4 1270t3o 42dr56i r15xk9h 2j33krk 253
be78 ac742a1 r15u9q5 zh86b2l ii286fq 9x482kt 6oa6m6u x3463ac i4l56nq g07h55q 0
81qc9t r159r1u"

# Define the target pattern
target_pattern = r"r15\w+"

# Use re.findall() to find the device IDs that match the target pattern
matching_device_ids = re.findall(target_pattern, devices)

# Display the results
print(matching_device_ids)

# This code will correctly find and display the device IDs that start with "r1
5" from the devices string.
```

```
['r151dm4', 'r15xk9h', 'r15u9q5', 'r159r1u']
```

Hint 1

Hint 2

Task 5

Now, the next task you're responsible for is analyzing a network security log file and determining which IP addresses have been flagged for unusual activity.

You're given the log file as a string stored in a variable named `log_file`. There are some invalid IP addresses in the log file due to issues in data collection. Your eventual goal is to use regular expressions to extract the valid IP addresses from the string.

Start by displaying the contents of the `log_file` to examine the details inside. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [7]: import re

        # Assign `log_file` to a string containing username, date, login time, and IP
        address for a series of login attempts
        log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
        6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
        022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
        \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
        192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
        12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
        \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
        2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
        05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

        # Display contents of `log_file`
        print(log_file)

        # Extract valid IP addresses using regular expressions
        valid_ip_addresses = re.findall(r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b', log_fil
        e)

        # Display the extracted valid IP addresses
        print("Valid IP Addresses:")
        print(valid_ip_addresses)

        # This code uses the re.findall function to find all occurrences of valid IP a
        ddresses in the log_file using a regular expression pattern. The extracted val
        id IP addresses are then printed.
```

```
eraab 2022-05-10 6:03:41 192.168.152.148
iuduike 2022-05-09 6:46:40 192.168.22.115
smartell 2022-05-09 19:30:32 192.168.190.178
arutley 2022-05-12 17:00:59 1923.1689.3.24
rjensen 2022-05-11 0:59:26 192.168.213.128
aestrada 2022-05-09 19:28:12 1924.1680.27.57
asundara 2022-05-11 18:38:07 192.168.96.200
dkot 2022-05-12 10:52:00 1921.168.1283.75
abernard 2022-05-12 23:38:46 19245.168.2345.49
cjackson 2022-05-12 19:36:42 192.168.247.153
jclark 2022-05-10 10:48:02 192.168.174.117
alevitsk 2022-05-08 12:09:10 192.16874.1390.176
jrafael 2022-05-10 22:40:01 192.168.148.115
yappiah 2022-05-12 10:37:22 192.168.103.10654
daquino 2022-05-08 7:02:35 192.168.168.144
Valid IP Addresses:
['192.168.152.148', '192.168.22.115', '192.168.190.178', '192.168.213.128',
'192.168.96.200', '192.168.247.153', '192.168.174.117', '192.168.148.115', '1
92.168.168.144']
```

Hint 1

Task 6

In this task, you'll build a regular expression pattern that you can use later on to extract IP addresses that are in the form of xxx.xxx.xxx.xxx. In other words, you'll extract all IP addresses that contain four segments of three digits that are separated by periods.

Write a regular expression pattern that will match with these IP addresses and store it in a variable named `pattern`. Use the regular expression symbols `\d` and `\.` in your pattern. Note that the symbol `\d` matches with digits, in other words, any integer between 0 and 9. Be sure to replace the `### YOUR CODE HERE ###` with your own code. Since you'll just build the pattern here, there won't be any output when you run this cell.

In [8]:
```
import re

# Assign `log_file` to a string containing username, date, login time, and IP
address for a series of login attempts
log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
\naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
\njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression pattern that will match with IP add
resses of the form xxx.xxx.xxx.xxx
pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'

# Display contents of `log_file`
print(log_file)

# Display the assigned pattern
print("Pattern for IP addresses:")
print(pattern)

# Extract valid IP addresses using the pattern
valid_ip_addresses = re.findall(pattern, log_file)

# Display the extracted valid IP addresses
print("Valid IP Addresses:")
print(valid_ip_addresses)
```

```
eraab 2022-05-10 6:03:41 192.168.152.148
iuduike 2022-05-09 6:46:40 192.168.22.115
smartell 2022-05-09 19:30:32 192.168.190.178
arutley 2022-05-12 17:00:59 1923.1689.3.24
rjensen 2022-05-11 0:59:26 192.168.213.128
aestrada 2022-05-09 19:28:12 1924.1680.27.57
asundara 2022-05-11 18:38:07 192.168.96.200
dkot 2022-05-12 10:52:00 1921.168.1283.75
abernard 2022-05-12 23:38:46 19245.168.2345.49
cjackson 2022-05-12 19:36:42 192.168.247.153
jclark 2022-05-10 10:48:02 192.168.174.117
alevitsk 2022-05-08 12:09:10 192.16874.1390.176
jrafael 2022-05-10 22:40:01 192.168.148.115
yappiah 2022-05-12 10:37:22 192.168.103.10654
daquino 2022-05-08 7:02:35 192.168.168.144
Pattern for IP addresses:
\b(?:\d{1,3}\.){3}\d{1,3}\b
Valid IP Addresses:
['192.168.152.148', '192.168.22.115', '192.168.190.178', '192.168.213.128',
'192.168.96.200', '192.168.247.153', '192.168.174.117', '192.168.148.115', '1
92.168.168.144']
```

Hint 1

Hint 2

Hint 3

Task 7

In this task, you'll use the `re.findall()` function on the regular expression pattern stored in the `pattern` variable and the provided `log_file` to extract the corresponding IP addresses. Afterwards, run the cell and take note of what it outputs. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [10]: import re

# Assign `log_file` to a string containing username, date, login time, and IP
address for a series of login attempts
log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
\naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
\njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression pattern that will match with IP add
resses of the form xxx.xxx.xxx.xxx
pattern = r"\d\d\d\.\d\d\d\.\d\d\d\.\d\d\d"

# Use the `re.findall()` function on `pattern` and `log_file` to extract the I
P addresses of the form xxx.xxx.xxx.xxx and display the results
extracted_ip_addresses = re.findall(pattern, log_file)

# Display the extracted IP addresses
print("Extracted IP Addresses:")
print(extracted_ip_addresses)

# This code uses the provided regular expression pattern and the re.findall()
function to extract IP addresses from the log_file and then prints the result
s.
```

```
Extracted IP Addresses:
['192.168.152.148', '192.168.190.178', '192.168.213.128', '192.168.247.153',
'192.168.174.117', '192.168.148.115', '192.168.103.106', '192.168.168.144']
```

Hint 1

Hint 2

Question 2

What are some examples of IP addresses that were extracted? What are some examples of IP addresses that were not extracted? Do any that were not extracted seem to be valid IP addresses?

Here's all the information i found within my code that shows examples of the extracted/non extracted ip addresses in detail:

Examples of Extracted IP Addresses (Valid):

'192.168.152.148' '192.168.22.115' '192.168.190.178' '192.168.213.128' '192.168.96.200' '192.168.247.153' '192.168.174.117' '192.168.148.115' '192.168.103.106' Examples of Non-extracted IP Addresses (Invalid):

'1923.1689.3.24' (Invalid: Segments have more than three digits) '1924.1680.27.57' (Invalid: Segments have more than three digits) '1921.168.1283.75' (Invalid: Segments have more than three digits) '19245.168.2345.49' (Invalid: Segments have more than three digits) '192.16874.1390.176' (Invalid: Segments have more than three digits) '192.168.168.144' (Invalid: Segments have more than three digits) Analysis:

Extracted IP addresses are in the valid format for examples like (192.168.152.148) and appear to be valid IPv4 addresses. Non-extracted IP addresses are invalid due to incorrect segment formats, values exceeding 255, or having more than four segments. In summary, the extracted IP addresses are valid IPv4 addresses, while the non-extracted ones are not due to various formatting issues that wont be compatible.

Task 8

There are some valid IP addresses in the `log_file` that you haven't extracted yet. This is because each segment of digits in a valid IP address can have anywhere between one and three digits.

Adjust the regular expression in the `pattern` to allow for variation in the number of digits in each segment. You can do this by using the `+` symbol after the `\d` symbol. Afterwards, use the updated `pattern` to extract remaining IP addresses. Then, run the cell to analyze the results. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [11]: import re

         # Assign `log_file` to a string containing username, date, login time, and IP
         address for a series of login attempts
         log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
         6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
         022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
         \naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
         192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
         12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
         \njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
         2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
         05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

         # Update `pattern` to a regular expression pattern that will match with IP add
         resses with any variation in the number of digits per segment
         pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'

         # Use the `re.findall()` function on `pattern` and `log_file` to extract the I
         P addresses and display the results
         extracted_ip_addresses = re.findall(pattern, log_file)

         # Display the extracted IP addresses
         print("Extracted IP Addresses:")
         print(extracted_ip_addresses)
```

```
Extracted IP Addresses:
['192.168.152.148', '192.168.22.115', '192.168.190.178', '192.168.213.128',
'192.168.96.200', '192.168.247.153', '192.168.174.117', '192.168.148.115', '1
92.168.168.144']
```

Hint 1

Hint 2

Question 3

What gets extracted here? Do all extracted IP addresses have between one and three digits in every segment?

Here were the following IP addresses that got extracted down below. I also, explain the details of the pattern structure for the IP addresses

Extracted IP Addresses:

'192.168.152.148' '192.168.22.115' '192.168.190.178' '192.168.213.128' '192.168.96.200' '192.168.247.153' '192.168.174.117' '192.168.148.115' '192.168.103.10654' Summary:

The updated pattern successfully extracts IP addresses with any variation in the number of digits per segment. All extracted IP addresses have between one and three digits in every segment, adhering to the allowed variation specified in the updated pattern for it to properly run effectively.

Task 9

Note that all the IP addresses are now extracted but they also include invalid IP addresses with more than three digits per segment.

In this task, you'll update the `pattern` using curly brackets instead of the `+` symbol. In regular expressions, curly brackets can be used to represent an exact number of repetitions between two numbers. For example, `{2,4}` in a regular expression means between 2 and 4 occurrences of something. Applying this to an example, `\w{2,4}` would match with two, three, or four alphanumeric characters. Afterwards, you'll call the `re.findall()` function on the updated `pattern` and the `log_file` and store the output in a variable named `valid_ip_addresses`.

Then, display the contents of `valid_ip_addresses` and run the cell to analyze the results. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [12]:
```
import re

# Assign `log_file` to a string containing username, date, login time, and IP
address for a series of login attempts
log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
\naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
\njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression that matches with all valid IP addr
esses and only those
pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'

# Use `re.findall()` on `pattern` and `log_file` and assign `valid_ip_addresse
s` to the output
valid_ip_addresses = re.findall(pattern, log_file)

# Display the contents of `valid_ip_addresses`
print(valid_ip_addresses)
```

```
['192.168.152.148', '192.168.22.115', '192.168.190.178', '192.168.213.128',
'192.168.96.200', '192.168.247.153', '192.168.174.117', '192.168.148.115', '1
92.168.168.144']
```

Hint 1

Hint 2

Question 4

What do you notice about the extracted IP addresses here compared to those extracted in the previous two tasks?

In the current task, the pattern used for extracting IP addresses (\b(?:\d{1,3}.){3}\d{1,3}\b) ensures that only valid IP addresses with one to three digits in every segment are extracted. This is achieved by specifying the allowed range of repetitions for each digit segment using curly brackets {1,3}. Compared to previous tasks, where variations in the number of digits were allowed, the extracted IP addresses in this task strictly adhere to the standard format of valid IPv4 addresses.

Task 10

Now, all of the valid IP addresses have been extracted. The next step is to identify flagged IP addresses.

You're given a list of IP addresses that have been previously flagged for unusual activity, stored in a variable named `flagged_addresses` . When these addresses are encountered, they should be investigated further. This list is just for educational purposes and contains examples of private IP addresses that are found only within internal networks.

Display this list and examine what it contains by running the cell. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [13]:
```
# Assign `flagged_addresses` to a list of IP addresses that have been previous
ly flagged for unusual activity
flagged_addresses = ["192.168.190.178", "192.168.96.200", "192.168.174.117",
"192.168.168.144"]

# Display the contents of `flagged_addresses`
print("Flagged IP Addresses:")
print(flagged_addresses)
```

```
Flagged IP Addresses:
['192.168.190.178', '192.168.96.200', '192.168.174.117', '192.168.168.144']
```

Hint 1

Task 11

Finally, you will write an iterative statement that loops through the `valid_ip_addresses` list and checks if each IP address is flagged. In the following code, the `address` will be the loop variable. Also, include a conditional that checks if the `address` belongs to the `flagged_addresses` list. If so, it should display `"The IP address ______ has been flagged for further analysis."` If not, it should display `"The IP address ______ does not require further analysis."` Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [14]:
```
import re

# Assign `log_file` to a string containing username, date, login time, and IP
address for a series of login attempts
log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:4
6:40 192.168.22.115 \nsmartell 2022-05-09 19:30:32 192.168.190.178 \narutley 2
022-05-12 17:00:59 1923.1689.3.24 \nrjensen 2022-05-11 0:59:26 192.168.213.128
\naestrada 2022-05-09 19:28:12 1924.1680.27.57 \nasundara 2022-05-11 18:38:07
192.168.96.200 \ndkot 2022-05-12 10:52:00 1921.168.1283.75 \nabernard 2022-05-
12 23:38:46 19245.168.2345.49 \ncjackson 2022-05-12 19:36:42 192.168.247.153
\njclark 2022-05-10 10:48:02 192.168.174.117 \nalevitsk 2022-05-08 12:09:10 19
2.16874.1390.176 \njrafael 2022-05-10 22:40:01 192.168.148.115 \nyappiah 2022-
05-12 10:37:22 192.168.103.10654 \ndaquino 2022-05-08 7:02:35 192.168.168.144"

# Assign `pattern` to a regular expression that matches with all valid IP addr
esses and only those
pattern = "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

# Use `re.findall()` on `pattern` and `log_file` and assign `valid_ip_addresse
s` to the output
valid_ip_addresses = re.findall(pattern, log_file)

# Assign `flagged_addresses` to a list of IP addresses that have been previous
ly flagged for unusual activity
flagged_addresses = ["192.168.190.178", "192.168.96.200", "192.168.174.117",
"192.168.168.144"]

# Iterative statement begins here
# Loop through `valid_ip_addresses` with `address` as the loop variable
for address in valid_ip_addresses:

    # Conditional begins here
    # If `address` belongs to `flagged_addresses`, display "The IP address ___
___ has been flagged for further analysis."
    if address in flagged_addresses:
        print(f"The IP address {address} has been flagged for further analysi
s.")

    # Otherwise, display "The IP address ______ does not require further analy
sis."
    else:
        print(f"The IP address {address} does not require further analysis.")

        # This code will iterate through the valid_ip_addresses list, check if
each IP address is flagged, and display the appropriate message based on wheth
er further analysis is required or not.
```

```
The IP address 192.168.152.148 does not require further analysis.
The IP address 192.168.22.115 does not require further analysis.
The IP address 192.168.190.178 has been flagged for further analysis.
The IP address 192.168.213.128 does not require further analysis.
The IP address 192.168.96.200 has been flagged for further analysis.
The IP address 192.168.247.153 does not require further analysis.
The IP address 192.168.174.117 has been flagged for further analysis.
The IP address 192.168.148.115 does not require further analysis.
The IP address 192.168.103.106 does not require further analysis.
The IP address 192.168.168.144 has been flagged for further analysis.
```

Hint 1

Hint 2

Hint 3

Conclusion

What are your key takeaways from this lab?

Key Takeaways:

1. **Regular Expressions (Regex):** Regular expressions are powerful tools for pattern matching in strings. They allow you to define patterns and search for matches within text.
2. **Validating IP Addresses:** Using regular expressions, you can define patterns for valid IP addresses. In this lab, the pattern `\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}` was used to match valid IPv4 addresses.
3. **Extracting Information:** The `re.findall()` function is useful for extracting information that matches a given pattern from a text string. In this lab, it was used to extract valid IP addresses from a log file.
4. **Conditional Statements:** When processing data, conditional statements play a crucial role. In the final task, an iterative statement was used to loop through valid IP addresses and check whether each address was flagged for further analysis.
5. **Data Analysis:** Analyzing log files, validating and extracting information, and making decisions based on conditions are common tasks in data analysis and cybersecurity.
6. **Iterative Statements:** Looping through lists or collections allows for efficient processing of data. In this lab, a `for` loop was used to iterate through the list of valid IP addresses.

Overall, this lab provided hands-on experience in using regular expressions for pattern matching, extracting information from log files, and making decisions based on the extracted data. These skills are valuable in various domains, including data analysis and cybersecurity. This lab provided useful information to grow heavier on my skillset.

```
In [ ]:
```